Semantic-based transactional support and recovery for nested composite software services

DESCRIPTION

CROSS-REFERENCE TO RELATED APPLICATION

[Para 1] This application claims the benefit of U.S. Provisional Application Serial No. 60/481,348, filed on September 10, 2003, entitled "Semantic-based transactional support and recovery for nested composite software services", which is incorporated herein by reference.

BACKGROUND OF THE INVENTION

[Para 2] 1. Field of the Invention

[Para 3] The present invention relates to the field of software development and, in particular, to service-oriented programming and transactional support for nested composite software services.

[Para 4] 2. Description of the Related Art

[Para 5] The reliable and correct execution of composite Web services is a key concern in the implementation of the emerging service-oriented systems. Addressing the reliability and correctness of nested composite Web services at runtime requires the addition of fault tolerance and transactional support to service-oriented platforms. Unfortunately, none of the known transaction processing technologies provides a model suitable and sufficient for the emerging service-oriented paradigm. This is mainly due to the diverse transactional behavior of heterogeneous and distributed Web service implementations.

[Para 6] The four main categories of existing transaction processing technologies address specific and limited semantic spaces:

[Para 7] **Database transactions**: This model involves transaction-processing technologies under full control of Data Base Management Systems (DBMS). These technologies assume a homogenous semantic space and dataset definition, based on relational database concepts, and thus are too rigid in definition. Furthermore, the transaction manager is non-extensible and closed to other transaction processing models.

[Para 8] Distributed Object transactions such as CORBA's Object Transaction Service (OTS) and X/Open DTP. Here, objects are only characterized by their interface and the transaction manager does not address the persistence of state. This means that the participating object (or the resource manager) must guarantee recoverability and durability by itself. Once the object provides the required interfaces and follows the specified Two-Phase-Commit (2PC) protocol, it can be used as part of transactions. While this provides a perfect heterogeneous model for automating atomic Web services used within the definition of composite Web services, it is demonstrably insufficient in addressing the transactional support of composite Web services for two main reasons: 1) The service-oriented platform that supports the implementation of composite Web services and provides transaction management capabilities must directly address the persistence and recovery of composite Web services. This is outside the semantic space for OTS and DTP. 2) The semantic space implied by these models strictly requires commit/rollback confirmation. This does not provide any flexibility in defining transactional composite Web services that may need to contain one or more Web services with optional, compensating, or alternative transactional behaviors (refer to the next section for an explanation of these behaviors).

[Para 9] **Component container-managed transactions (such as Sun's EJB)**: This model mainly follows the distributed object model above and thus suffers from the same limitations with respect to the semantic needs of composite Web services. A point worthy of mention is that this model, unlike the distributed object transaction model, uses declarative transaction management. This effectively simplifies the coding of transactions since it only requires setting transactional attributes to adjust a component's behavioral

operations. This technique is inspiring in that it separates transaction behavior from business logic, thus allowing reusability. However, it is limited in that it does not provide a way for the external client to override these internal attributes.

[Para 10] **Message-oriented transactions and transactional publish/subscribe:** These transactions group a set of messages, a unit of work, with strict "all-or-nothing" semantics without addressing data processing and the consistency of data transmissions. This transaction model, like the others mentioned, does not provide semantics that address nested, heterogeneous transactions.

[Para 11] Undoubtedly, the definition of a new semantic space is required for adding reliability and correctness to nested composite Web services. Intuitively speaking, the new semantic space must blend ideas from various existing models to provide the flexibility required for accommodating behavioral diversity demanded by composite systems. The present invention is at the forefront of defining this new semantic space for composite systems.

[Para 12] As the first step in adding reliability to composite Web services we define the semantics of the Web service dataset, herein below referred to as a service-set, that needs to be persistent for the purpose of state recovery after any failure. A service-set is composed of four distinct, and possibly empty, data-elements: 1) the inputs, 2) the properties (such as the URL of the service), 3) the outputs, and 4) the composite-set, where the composite-set is either the empty dataset (in the case of atomic Web services), or a set containing one or more service-sets. Note that the recursive definition of the nested-set element allows us to later accommodate recovery of nested composite Web services to any depth.

[Para 13] The recovery of a composite Web service will, at times, require re-execution of embedded services that were successfully executed before the crash. For example, consider a time-sensitive Web service such as GetStockQuote that takes a ticker symbol as input and returns the current value of the stock associated with that ticker as output. Imagine that GetStockQuote is embedded within a composite service that crashes sometime after the execution of the GetStockQuote service. Here, upon recovery, instead

of simply restoring the output element of the persistent service-set associated with GetStockQuote, the correct behavior may be to re-execute GetStockQuote to avoid stale and out-of-date data. As a contrary example, consider an IncrementCounterAndGetValue Web service that returns the value of a counter after incrementing it by one. Imagine that this service is embedded in a composite service that crashes sometime after the successful execution of IncrementCounterAndGetValue. Here, unlike in the GetStockQuote example, upon recovery of the parent composite service, IncrementCounterAndGetValue must not be re-executed. Instead, the output of the IncrementCounterAndGetValue instance that was executed prior to the crash must be restored from the output element of the associated service-set. These two contrary examples are intended to demonstrate that not only is the definition of a Web service dataset specific to a service-oriented paradigm, but also the behavior of recovery depends on the application of the service. To address diverse recovery behavior for services, we will introduce a declarative service-set definition where the behavior of recovery can be selected from a finite set of possibilities, decided at the application level, and set as a property on the service-set.

[Para 14] A semantic space for defining the transactional behavior of Web services requires the same type of flexibility as that required for their recovery. It is desirable to define transactional composite Web services without requiring that all embedded services, to any depth, must be transactional services. For example, one may want to define a transactional composite Web service that embeds the non-transactional GetStockQuote service. In other words, the semantic space for transactional Web services must allow for optional, or non-transactional, Web services to be embedded within transactional composite Web services.

[Para 15] Similarly, to accommodate the diverse transactional behavior of heterogeneous Web service implementations, this semantic space must allow for compensation of embedded Web services upon rollback of the parent Web service. For example, upon rollback of a parent composite Web service that embeds a SendEmail Web service, the designer of the parent service must be

able to compensate for the already executed SendEmail. In this case, another SendEmail service may be executed to notify the recipient of the cancellation of the first service. Another behavior that must be accommodated in the new space is alternative Web service routing instead of rollback. For example, consider a child ShipThroughFedEx Web service that triggers a failure/rollback upon execution. Here, the designer of the parent service must be able to instruct the transactional system to route to an alternative ShipThroughDHL service instead of rolling back the other prepared services within the parent service.

[Para 16] A flexible semantic space for Web service transactions must also allow combining internal as well as external handling of transactional behavior for Web services. For example, it should be possible to embed a transactional Web service within a composite Web service and then override the rollback behavior of the embedded service to invoke an alternative service, rather than rolling back. For atomic Web services, semantics similar to container-manager, or distributed object transaction model can be leveraged to accommodate automated commit/rollback behavior internal to the atomic Web service.

[Para 17] 3. General Background

[Para 18] A software service, or service for short, including but not limited to a Web service, is a discrete software task that has a well-defined interface and may be accessible over the local and/or public computer networks or maybe only available on a single machine. Web services can be published, discovered, described, and accessed using standard-based protocols such as UDDI, WSDL, and SOAP/HTTP.

[Para 19] A software service interface, in concept, represents the inputs and outputs of a black-boxed software service as well as the properties of that service, such as name and location. Take, for example, the interface of a simple software service named GetStockQuote, which retrieves simple stock quote information [FIGURE 1]. This service takes a ticker symbol input and returns the last trade price amount as well as some additional stock quote details, such as the day high and day low. Note that in order to use, or consume, a service, only knowledge of its interface is required. This means

that as long as the interface of a service remains the same, different implementations of the service can be swapped in and out without affecting its consumers. This, as well as the fact that a service is a language- and platform-neutral concept, is one of the keys to the flexibility of service-oriented architectures.

[Para 20] An atomic service is a software service that is implemented directly by a segment of software code. In the existing NextAxiom™ HyperService™ Platform, atomic Web services are dispatched via a library. A library is a light, language- and platform-neutral wrapper that is linked to one or more atomic Web service implementations. Atomic Web services are logically indivisible Web services that represent "raw materials" to the HyperService™ platform.

[Para 21] A composite service is a software service that consumes any number of other atomic or composite services. In the HyperService™ platform, a composite Web service is implemented with a metadata-driven model that is automatically interpreted by a high-performance run-time engine.

[Para 22] Visual metadata models, which represent composite software services implementations to the HyperService™ system, are created in a graphical, design-time environment and stored as XML models. This environment offers a new and powerful visual modeling paradigm that can be leveraged to enable the visual modeling of transactional behavior. This environment was specifically designed to enable collaborative, on-the-fly creation of software services by business process analysts or functional experts, who understand the business logic and application required to implement real-world business processes and applications, but have no knowledge of programming paradigms or Web service protocols. FIGURE 2 captures the implementation of a composite software service named "FindFiles".

[Para 23] Any software service that is consumed by a composite service model is said to be "nested" or "embedded" within that composite service. FIGURE 3 depicts a hypothetical composite service. This software service is composed of other composite services that can run locally or distributed.

SUMMARY OF THE INVENTION

[Para 24] A principle object of the present invention is to provide a system and a method for defining and automating service recovery and transactional behavior, including commit/rollback and compensation, of a software service, including, but not limited to, "Web services", where the software service maybe a composite software service, one that is composed of other services nested without a fixed limit on the depth of composition.

[Para 25] The present invention provides a declarative system where a software service can be declared as transactional or non-transactional and the said declaration is associated to its interface definition or associated to the context of the composite service that it is contained in, where furthermore, the declaration within the context can override the declaration within the interface. The said declarative system provides for customizing the behavior of the services contained in a composite service upon automatic recovery.

[Para 26] Furthermore, the present invention provides the ability to declare a contained service as transactional whether or not the containing composite service is marked as transactional.

[Para 27] A further object of the present invention is to provide a framework for extending the automated transactional behavior of the system through a plug-in architecture for non-composite services. The present invention provides a set of generic built-in database access services (e.g. update, query, delete, and add), that are non-composite services in nature, with automated transactional behavior that can be used within composite services using a composition tool.

[Para 28] The present invention provides a semantic-based means for defining the transactional and recovery behavior of optionally nested, composite software services, with respect to the context of the containing service, using a service composition tool.

[Para 29] Yet another object of the present invention is to provide a persistent, nested context mechanism capable of remembering the exact state

of execution corresponding to the invocation of nested composite services as it relates to transactional support. This context mechanism includes the shared memory used within a composite service as well as the context for non-composite services used within a composite service.

[Para 30] Other objects and advantages of this invention will be set in part in the description and in the drawings which follow and, in part, will be obvious from the description, or may be learned by practice of the invention. Accordingly, the drawings and description are to be regarded as illustrative in nature, and not as restrictive.

[Para 31] To achieve the forgoing objectives, and in accordance with the purpose of the invention as broadly described herein, the present invention provides methods, frameworks, and systems for defining and automating transactional behavior, including commit/rollback and compensation, of a software service. In preferred embodiments, this technique comprises: the association of commit/rollback and compensation ports to the graphical presentation of software service, contained within a composite service and manipulated through a composition tool; a plug-in framework for providing software wrappers around non-composite software service implementations to provide automatic transactional support behavior for the non-composite services contained within a composite service; a persistent context mechanism based on a directed execution graph and an invocation map associated to each composite instance; a system for automating the flow of execution based on user-defined semantic definitions and built-in behavior, as it relates to, commit, rollback and compensation of services within a composite service; a technique for communicating the context of each service, based on the definition of the services, including the commit, rollback and compensation by the module implementing the non-composite service as well as automated commit/rollback behavior for the shared memory used within the context of the nested composite services.

[Para 32] The present invention will now be described with reference to the following drawings, in which like reference numbers denote the same element throughout. It is intended that any other advantages and objects of the present

invention that become apparent or obvious from the detailed description or illustrations contained herein are within the scope of the present invention.

DESCRIPTION OF THE DRAWINGS

[Para 33] FIGURE 1 depicts an example interface of a software service named 'GetStockQuote'.

[Para 34] FIGURE 2 shows visual composition of a software service using a service composition and assembly tool, HyperService™ Studio.

[Para 35] FIGURE 3 shows the nesting of other software services within a composite software service.

[Para 36] FIGURE 4 shows an example of visual rollback and compensation ports on a software service defined within a composite implementation.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

[Para 37] The present invention provides a semantic space, as a visual interface, for defining transactional and recovery behavior within the implementation of composite software service, including, but not limited to Web services. This invention also provides a framework for a fault-tolerant, automated, nested, transactional system supporting the visual semantic space.

[Para 38] Assuming a visual software service composition and assembly tool, such as HyperService™ Studio with a snapshot of a definition in FIGURE 2, and an automatic flow platform, such as the HyperService™ platform, available at www.nextaxiom.com, we will now describe how such a composition tool and platform can be extended to accommodate declarative semantic-based recovery and transactional support for nested composite web services.

[Para 39] A context sensitive attribute is added to the properties of a service contained in a composite service declaring the behavior of the service node

upon recovery in the event of system crash or other similar abnormal termination events. This attribute indicates whether to automatically re-invoke the service instance corresponding to the definition even if the persistent state of the execution of the containing composite service indicates that a corresponding service instance was already invoked successfully prior to the system crash.

[Para 40] To accommodate a declarative semantic-based behavior for nested composite services, a set of attributes and a visual flow port(s) are added to a composite service definition and the contained service nodes within a composite service definition. An attribute associated to a service interface definition, or the composite definition allows the application developer, or any user of the composition tool, to indicate whether a transactional context should be created for the composite instance associated to the composite definition upon execution. In other words, what is declared is, whether the composite definition is transactional. Similar attribute is associated to a service node contained within a composite definition to indicate whether the contained service, in the context of the containing service, is transactional. This allows the application developer to overwrite the respective behavior of a service that is declared on its interface or definition based on the context of its usage within another composite service.

[Para 41] Next, one or more flow ports, refer to FIGURE 4, are added to the visual presentation of each service node within a persistent composite service definition. Each port allows the application developer to determine the flow of service invocation when a rollback/compensation event is triggered; for example, as the result of the failure of one of the contained services.

[Para 42] The flow port allows the application developer to route the flow of invocation to another service or programming construct within the composite service definition. An attribute on the port, allows the application developer to declare whether the built-in commit/rollback/compensate behavior of the associated service should be extended through the connectivity of the rollback port, or should be overwritten.

[Para 43] Another behavior of the flow port, which appears on the visual presentation of non-composite contained services with libraries not supporting the commit/rollback/compensate interface later discussed as part of the present invention, allows the application developer to define a flow of invocation that can compensate for the overall failure of the containing composite service that triggered the activation of this port for the purpose of compensation.

[Para 44] To support the automatic implementation of transactional behavior for atomic Web services, a wrapper interface must be implemented for all service libraries wanting to provide transactional support at the atomic level. This transactional interface provides for the implementation of internal 2PC or automated compensation depending on the type of underlying system, delegated to the engineer of the library. One of the methods on the library interface for transactional support, allows the automating platform to request a unique context from the library. Upon the invocation of a transactional composite service, a unique context is acquired from each of the libraries associated with the services contained within that composite service. Another method on the wrapper interface, enables the automation platform to prepare a service associated to the implementing library within the unique context obtained from the library. Yet another method of the wrapper interface, enables the automation platform to commit or rollback a library context when needed.

[Para 45] As a non-exclusive example, a library implementation for a set of atomic database access services is provided by the automating platform. In this case, the unique context returned by the library, upon the request of the automating platform through the wrapper interface, is an object with a reference to a database connection. Every time the automating platform encounters a database access service associated to this library within the nested context of a composite service, regardless of the depth of nesting as long as one composite service within another is also marked as transactional, the automating platform calls the perform method with the associated unique context from the wrapper interface and as a result the database access library

passes the corresponding SQL query through the connection to the backend database without performing a commit. When the automating platform decides that the invoked database access service must be committed, that is usually when the root transactional composite service is ready to exit with success, then the automating platform calls the commit method with the unique context as a parameter. However, if the automating platform decides that the database access services must be rolled back, due to the need to rollback the containing composite service, the automating platform calls the rollback method of the wrapper interface with the associated context as a parameter. In this way the database access library discards the connection associated to the context after rollback the database transactions associated to that connection.

[Para 46] In providing the automatic implementation of the declarative behavior, a persistent, nested context mechanism is devised. This context mechanism uses a set of service interfaces to provide persistence for the state of the service-sets corresponding to the invoked services, and the overall state of invocation of a composite service, containing other services, nested to any level of depth. The use of a set of service interfaces for persisting the complete state of the invocation increases the flexibility and customizability of the system by providing a layer of encapsulation that allows the automating system to use different storage medium such as direct file system vs. relational database.

[Para 47] To create a context for a composite service first a directed graph, hereon referred to as the execution graph, is created that captures the dependencies of all the contained service nodes, based on the connectivity of services from the existing success/failure and the added commit/rollback/compensation ports. Then, an object, hereon referred to as the Invocation Map is created, based on the execution graph, to hold the context of a composite service upon invocation. Additionally, modifications to shared data structures accessed within the definition of the corresponding composite service are stored in the Invocation Map as an object holding the context of the shared memory and providing commit/rollback methods for

reflecting the changes, or discarding the changes done to shared memory within the context of a composite service instance.

[Para 48] Upon the invocation of a composite service, the automation platform, instantiates an Invocation Map, corresponding to the execution graph created based on the dependency of contained services. Then, it determines the set of services that can be invoked next by: 1) traversing the execution graph in the order of node dependencies with the nodes having no dependencies or dependencies to the input data of the composite service only, as the first set of nodes; and, 2) the nodes with dependencies to prior nodes and the inputs of the composite service, as the second set of nodes, and so on.. After preparing the input data for those services, the automation platform stores the corresponding service-sets in the Invocation Map, while the Invocation Map ensures the persistence of the service-set and its state. Then, the automation platform invokes the prepared services, while a unique key is associated to the invocation of each service, for the purpose of recovery from an unknown state. Hereon, we will refer to the invocation of each set of prepared services as a HyperCycle. Also, the Invocation Map holds the unique context associated to each atomic library, keyed by the identifier of that library, as part of its overall context.

[Para 49] To provide the automatic runtime behavior for transactional support, under the declarative and semantic-based visual techniques discussed in the present invention, the automation platform starts the invocation of a composite service instance, declared as transactional, by creating a persistent instance of an Invocation Map based on the composite service definition. Then, once services within each HyperCycle are invoked, if all the services are consumed successfully (or, even if some services failed, by the had a defined routing upon failure that was successful) and if the containing composite service is a root service (i.e. one that is contained within another composite service), the nested contained composite services and all their nested composite services to the depth of nesting, within the containing composite service, are committed together with the entire unique context associated to each library, if any, and the context associated to the shared

memory. If all the services contained within a composite service execute successfully (or, even if some services failed, by the had a defined routing upon failure that was successful), but the containing service is not a root containing service, the contained composite service will wait until it receives a commit or rollback call from the context of the containing service. Furthermore, if any of the services within a HyperCycle fai,l without a predefined routing upon failure, all the context within the Invocation Map of the composite service instance will be rolled back. This rollback will result in nested rollback, if any of the contained services are composite services, and it includes the rollback of shared memory and all the associated atomic libraries as discussed earlier. Once an invocation Map enters a rollback or a commit state, the connectivity of the rollback/compensation/commit ports of the contained services are implemented as follows: 1) If a flow port marked for rollback is connected, and if the port is declared as an overwriting port, the built-in rollback of the underlying service instance will not be requested by the automation platform; instead, the services connected to that port are invoked as the rollback act; 2) however, if the port is declared as an extending port, then after the completion of the rollback of the underlying service, the services connected to the port are invoked to extend the act of rollback. Upon rollback, if the compensation port of a service is connected to other services, those services are invoked by the automation platform . The act of rollback, is performed in a natural back tracking order with respect to all the contents of a composite service.

[Para 50] Incremental check-point recovery techniques, in conjunction with the persistent service-sets within the nested context mechanism and the declarative information associated to the definitions and interfaces, introduced as part of the present invention, can be used to automate the recovery of nested composite services. Basically, the persistent Invocation Map is used to store the status of the last service invoked within each HyperCycle. Upon a system failure that results in automatic recovery for the composite services declared as recoverable, The persistent Invocation Map associated to each instance of the composite service running at the time of the crash, is recovered. First, all the services that were successfully invoked prior to the

crash are examined. Any of those service are declared as time-sensitive services that require re-invocation upon recovery are re-invoked and their service-sets are modified to reflect the new results of invocation. Than, the invocation continues from the last HyperCycle under invocation at the time of the crash. When the invocation status of a contained service is within the last HyperCycle is unknown, the automated recovery platform, re-invokes the service with the same unique id used upon invocation prior to the crash; if the service is an atomic service, guarantying the only once invocation of the service is delegated to the implementing library. If the unknown state is associated to a composite service, the automation platform, will recognize the service request as a prior request through its id, and initiates the recovery of the state of that service, if not yet recovered, and returns the resulting service-set including the output and the state of execution for that service.